Real Time Systems1

lecture 8

Resource Access Protocols

The Priority Inversion Phenomenon

Resource Access Protocols

- A *resource* is any software structure that can be used by a process to advance its execution.

- Typically, a resource can be a data structure, a set of variables, a main memory area, a file, or a set of registers of a peripheral device

Resource Access Protocols

- A resource dedicated to a particular process is said to be *private*,
- whereas a resource that can be used by more tasks is called a *shared resource*.

- A shared resource protected against concurrent accesses is called an *exclusive resource*

Resource Access Protocols

- To ensure consistency of the data structures in exclusive resources, any concurrent operating system should use appropriate resource access protocols to guarantee a mutual exclusion among competing tasks.

- A piece of code executed under mutual exclusion constraints is called a *critical section*

Resource Access Protocols

$X=5$

Task1

$X=X+4$

Task2

$X=X-2$

Resource Access Protocols

- Any task that needs to enter a critical section must wait until no other task is holding the resource.

- A task waiting for an exclusive resource is said to be ***blocked*** on that resource, otherwise it proceeds by entering the critical section and holds the resource.

Resource Access Protocols

- When a task leaves a critical section, the resource associated with the critical section becomes *free*, and it can be allocated to another waiting task, if any.

- Operating systems typically provide a general synchronization tool, called a *semaphore* [Dij68, BH73, PS85], that can be used by tasks to build critical sections

Resource Access Protocols

- A semaphore is a kernel data structure that apart from initialization, can be accessed only through two kernel primitives, usually called ***wait*** and ***signal***.
- When using this tool, each exclusive resource *Rk* must be protected by a different semaphore *Sk* and each critical section operating on a resource *Rk* must begin with a ***wait*(*Sk*)** primitive and end with a ***signal*(*Sk*)** primitive

Resource Access Protocols

- All tasks blocked on a resource are kept in a queue associated with the semaphore that protects the resource.
- When a running task executes a ***wait*** primitive on a locked semaphore, it enters a *waiting* state, until another task executes a ***signal*** primitive that unlocks the semaphore.
- When a task leaves the waiting state, it does not go in the running state, but in the ready state, so that the CPU can be assigned to the highest priority task by the scheduling algorithm

Resource Access Protocols

- Here, we describe the main problems that may arise in a uniprocessor system when concurrent tasks use shared resources in exclusive mode, and we present some resource access protocols designed to avoid such problems and bound the maximum blocking time of each task.
- We then show how such blocking times can be used in the schedulability analysis to extend the guarantee tests derived for periodic task sets.

Resource Access Protocols
The Priority Inversion Phenomenon

- Consider two tasks $\tau 1$ and $\tau 2$ that share an exclusive resource *Rk* (such as a list) on which two operations (such as *insert* and *remove*) are defined.

- To guarantee the mutual exclusion, both operations must be defined as critical sections. If a binary semaphore *Sk* is used for this purpose, then each critical section must begin with a ***wait(Sk)*** primitive and must end with a ***signal(Sk)*** primitive

Resource Access Protocols
The Priority Inversion Phenomenon

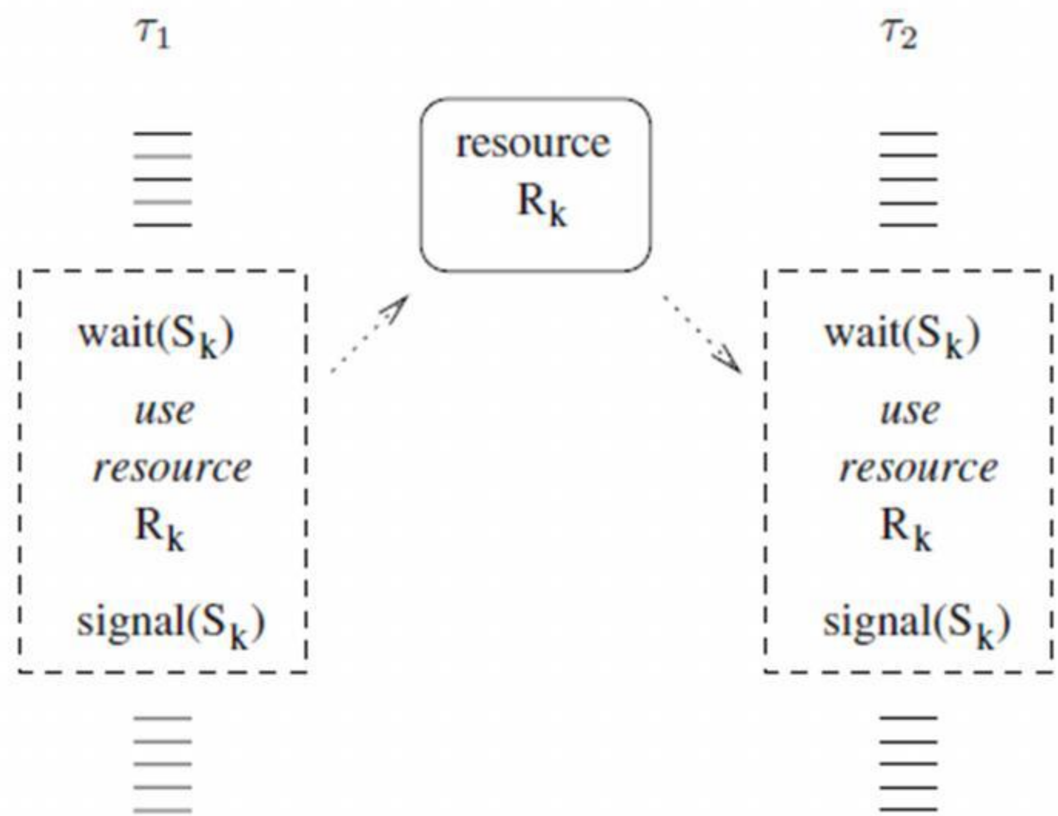

Figure 7.2 Structure of two tasks that share an exclusive resource.

Resource Access Protocols
The Priority Inversion Phenomenon

- If preemption is allowed and $\tau 1$ has a higher priority than $\tau 2$, then $\tau 1$ can be blocked in the situation depicted in Figure 7.3.

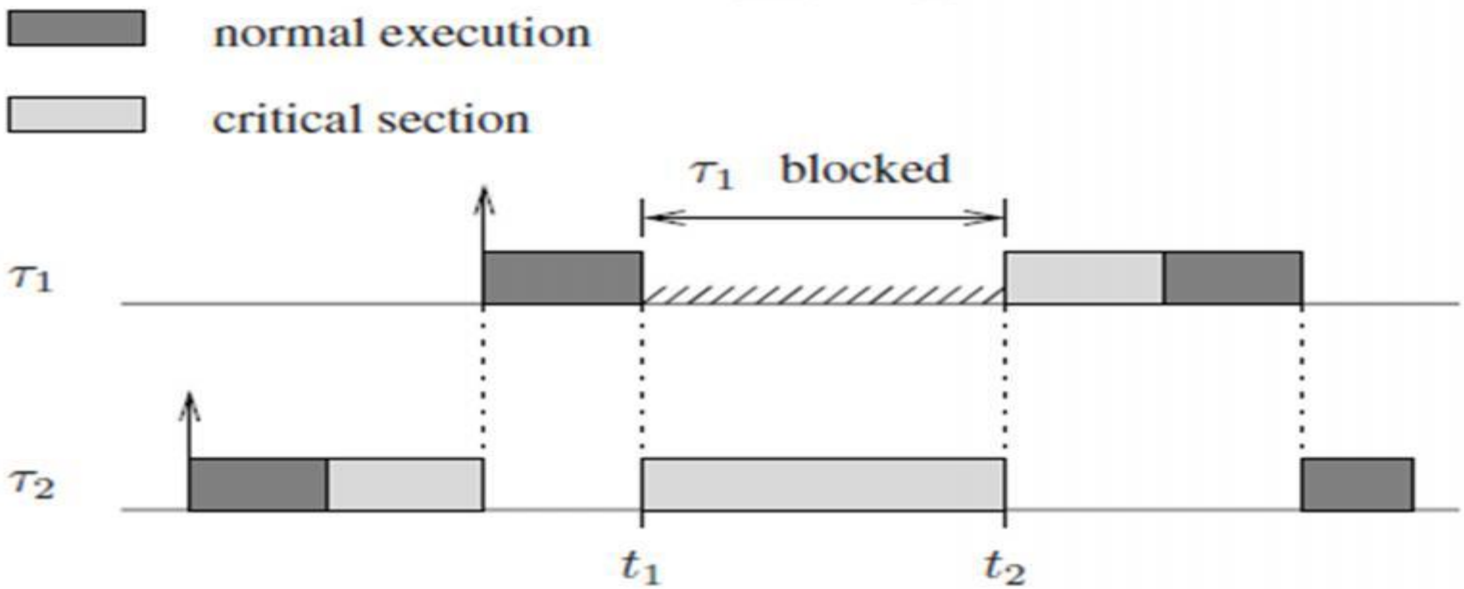

Figure 7.3 Example of blocking on an exclusive resource.

Resource Access Protocols
The Priority Inversion Phenomenon

- Here, task $\tau 2$ is activated first, and after a while, it enters the critical section and locks the semaphore. While $\tau 2$ is executing the critical section, task $\tau 1$ arrives, and since it has a higher priority, it preempts $\tau 2$ and starts executing.
- However, at time $t1$, when attempting to enter its critical section, $\tau 1$ is blocked on the semaphore, so $\tau 2$ resumes. $\tau 1$ has to wait until time $t2$, when $\tau 2$ releases the critical section by executing the *signal(Sk)* primitive, which unlocks the semaphore.

Resource Access Protocols
The Priority Inversion Phenomenon

- In this simple example, the maximum blocking time that τ 1 may experience is equal to the time needed by $\tau 2$ to execute its critical section.

- Such a blocking cannot be avoided because it is a direct consequence of the mutual exclusion necessary to protect the shared resource against concurrent accesses of competing tasks.

Resource Access Protocols
The Priority Inversion Phenomenon

- Unfortunately, in the general case, the blocking time of a task on a busy resource cannot be bounded by the duration of the critical section executed by the lower-priority task.

- In fact, consider the example illustrated in Figure 7.4. Here, three tasks $\tau 1$, $\tau 2$, and $\tau 3$ have decreasing priorities, and $\tau 1$ and $\tau 3$ share an exclusive resource protected by a binary semaphore *S*.

Resource Access Protocols
The Priority Inversion Phenomenon

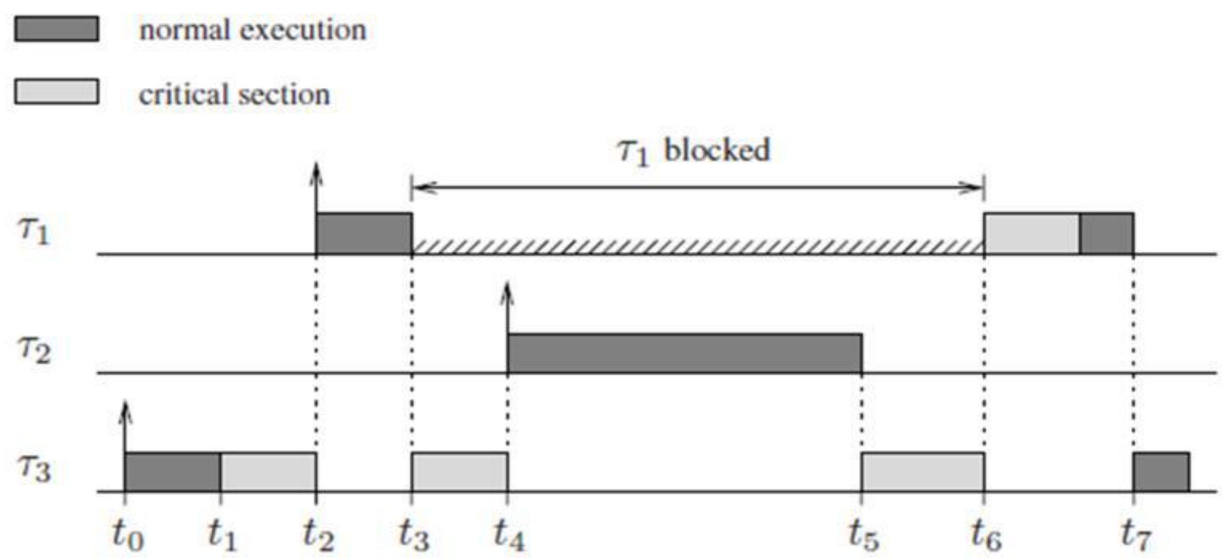

Figure 7.4 An example of priority inversion.

If $\tau 3$ starts at time $t0$, it may happen that $\tau 1$ arrives at time $t2$ and preempts $\tau 3$ inside its critical section. At time $t3$, $\tau 1$ attempts to use the resource, but it is blocked on the semaphore S; thus, $\tau 3$ continues the execution inside its critical section.

Resource Access Protocols
The Priority Inversion Phenomenon

- Now, if $\tau 2$ arrives at time $t4$, it preempts $\tau 3$ (because it has a higher priority) and increases the blocking time of $\tau 1$ by its entire duration.

- As a consequence, the maximum blocking time that $\tau 1$ may experience does depend not only on the length of the critical section executed by $\tau 3$ but also on the worst-case execution time of $\tau 2$!

Resource Access Protocols
The Priority Inversion Phenomenon

- This is a situation that if it recurs with other medium-priority tasks, can lead to uncontrolled blocking and can cause critical deadlines to be missed.

- A *priority inversion* is said to occur in the interval [$t3$, $t6$], since the highest-priority task $\tau 1$ waits for the execution of lower priority tasks ($\tau 2$ and $\tau 3$).

Resource Access Protocols
The Priority Inversion Phenomenon

- In general, the duration of priority inversion is unbounded, since any intermediate-priority task that can preempt $\tau3$ will indirectly block $\tau1$. Several approaches have been defined to avoid priority inversion, both under fixed and dynamic priority scheduling.

- All the methods developed in the context of fixed priority scheduling consist in raising the priority of a task when accessing a shared resource, according to a given protocol for entering and exiting critical sections